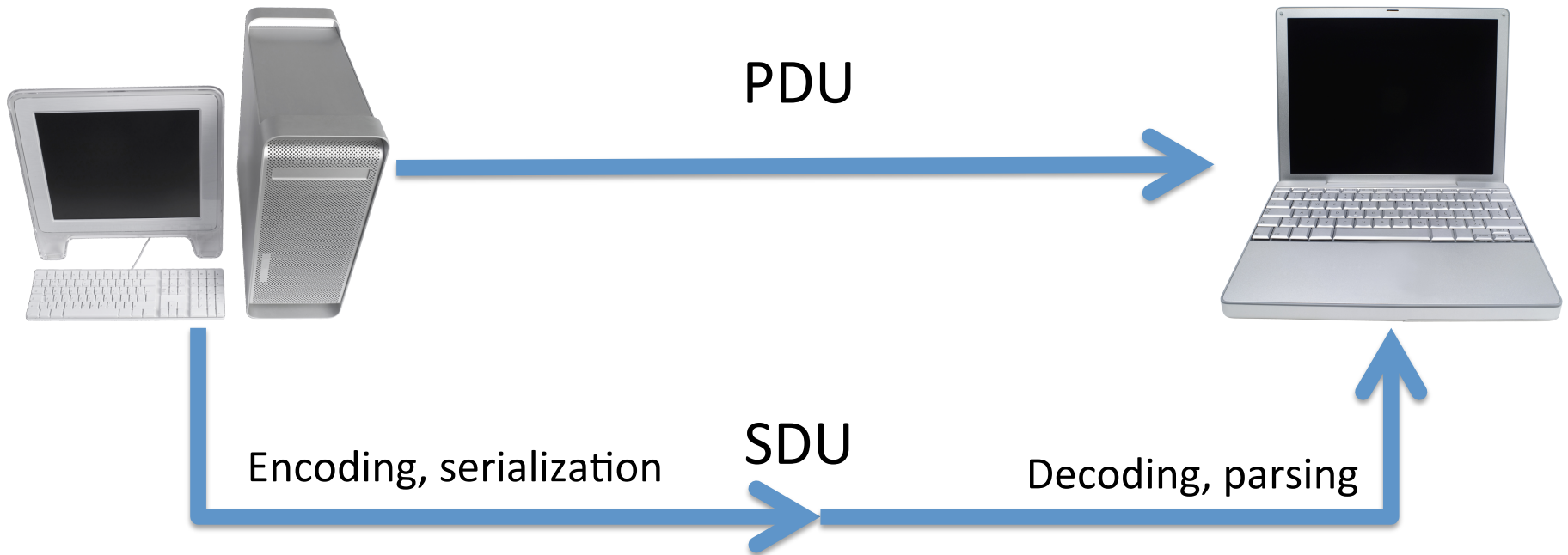


Data modeling, data encoding

What we want to do



- Get complex data structure from one node to another

Data transfer

- Data transfer can be
 - Binary
 - ASN.1
 - Mostly in telecommunications networks and lower layer protocols
 - Character based
 - XML, JSON, YAML, CSV
 - In the Internet infrastructure and higher layer protocols
- Data can be sent in
 - One large PDU
 - Several chunks

Data transfer

- Optimal data size
 - t : message overhead (flow control, address, checksum) size
 - d : useful data size
 - a : acknowledge message size
 - p_d, p_a : probabilities of bit errors in data and ack messages
- If $p_d=p_a=0$, d can be arbitrarily large
 - work over TCP

Data transfer

- If $p_d > 0$ and $p_a > 0$
 - The probability of retransmission is:
$$p_r = 1 - (1 - p_d)(1 - p_a)$$
 - The probability that i retransmissions are needed is: $p_i = (1 - p_r)p_r^{i-1}$
 - Expected number of retransmissions is:
$$R = 1/(1 - p_r)$$
 - Efficiency is: $E = d/R(d+t+a)$
 - The optimum data size can be found by solving:
$$\delta E / \delta d = 0$$

Data transfer

- When to use chunks:
 - Unreliable medium
 - Lower layer protocols
- When to use one complex PDU:
 - Reliable medium
 - Applications operating over higher layer protocols (TCP, HTTP)

What we know about the data

- Mandatory fields
 - Fields without initial value in the function signature
- Optional fields
 - Fields with initial value in the function signature
- Extra fields
 - Fields present, but not specified in the requirements
 - Must be ignored, must not cause error
- Ordering of fields may or may not be defined

Data modeling

- Data modeling know in:
 - Database theory – Entity Relationship Diagrams
 - Object oriented modeling – UML Class Diagrams
 - These have about the same expressive power
- Data modeling has several abstraction levels:
 - Conceptual model – lots of informal elements
 - Logical model
 - Physical model – formal
- For encoding/decoding we need formal models

Problems

- Data modeling describes relational models
- Data serialization produces hierarchical models, hash trees
- What to do with many-to-many relationships?
 - This is the main cause of data structure refactorization

XML

- Describes a piece of data (a value) that can be exchanged between systems
- Primarily used in SOA and web based applications
- Provides structural relationship information, metadata
- Semantics can be assigned to its elements: XSD
- Appearance can be assigned to its elements: XSL

XML

- Components of an XML document
 - Declarations
`<?xml version="1.0" encoding="UTF-8"?>`
 - Tags (start, end)
`<data>`, `</data>`, `<data />`
 - Attributes
`id="42"`
 - Data
Some piece of text
 - Elements
`<data id="42">Some piece of text</data>`
 - Comments
`<!-- A comment -->`

XML

- XML is hierarchical
 - An element may contain other elements or character data (text nodes)
- XML is good at representing
 - Lists
 - Trees
 - One-to-one and one-to-many relations
- XML is bad at representing
 - Many-to-many relations
 - Binary data

XML format, syntax

- Well-formed to be parsed successfully
 - XML declaration
 - Exactly one root element
 - Each element is properly nested
 - Attributes are in quotes
 - Special characters (' " & < > :) escaped
 - No spaces, special characters in element names
- XML is
 - Character based, any character encoding, default is UTF-8
 - Case sensitive

XML semantics

- Many XML documents can use the same element names
- To avoid semantic collisions: namespaces
 - An URN is assigned to a namespace URI
 - Elements tagged with the URN
 - Namespace can be introduced at any level of the XML hierarchy

XML Schema

- Defines the formal structure of an XML document (value), message format of an SDU
- Alternatives: DTD, SOX
- XSD
 - Written in XML
 - Automatic schema creation
 - Self-documentation
 - Provides semantics

XML Schema

- Defines:
 - Elements, attributes
 - Parent child relations of elements
 - Relation cardinality
 - Text nodes, empty nodes
 - Text node type
 - Default elements, attributes

XML Schema

- Root element

```
<?xml version="1.0"?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://agile.tmit.bme.hu"
xmlns="http://agile.tmit.bme.hu"
elementFormDefault="qualified">
```

```
...
```

```
</xs:schema>
```

- xmlns:xs: XML Schema types
- targetNamespace: element semantics
- xmlns: default namespace
- elementFormDefault: if tag must be qualified with the namespace

XML Schema

- A value of a type

```
<xs:element name="i" type="xs:integer"/>
```

- Built-in primitive types: xs:integer, xs:string, xs:date, xs:time, xs:datetime, xs:boolean, xs:float
- Default value: default attribute
- Constant: fixed attribute
- Cardinality: minOccurs, maxOccurs, optional if minOccurs is 0

XML Schema

- Attribute

```
<xs:attribute name="i" type="xs:integer"/>
```

- Must be enclosed in an element
- Similar to elements

XML Schema

- **Complex type**

```
<xs:complexType name="T">  
  <xs:sequence>  
    <xs:element name="i" type="xs:integer"/>  
    <xs:element name="d" type="xs:datetime"/>  
    <xs:element name="s" type="xs:string"/>  
  </xs:sequence>  
</xs:complexType>
```

- **Structure: sequence, union: choice, set: all**
- **Type reference**

```
<xs:element name="E" type="T"/>
```

XML Schema

- Type specialization, restrictions

```
<xs:simpleType name="T">  
  <xs:restriction base="xs:integer">  
    <xs:minInclusive value="1"/>  
    <xs:maxInclusive value="10"/>  
  </xs:restriction>  
</xs:simpleType>
```

- Enum

```
<xs:simpleType name="T">  
  <xs:restriction base="xs:integer">  
    <xs:enumeration value="A"/>  
    <xs:enumeration value="B"/>  
  </xs:restriction>  
</xs:simpleType>
```

- Pattern based restriction

XML Schema

- Type generalization

```
<xs:complexType name="T2">  
  <xs:complexContent>  
    <xs:extension base="T">  
      <xs:sequence>  
        ..  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>
```

- simpleContent and complexContent

XML Schema

- Global vs. nested types

```
<xs:complexType name="T">
```

```
...
```

```
</xs:complexType>
```

```
<xs:element name="E" type="T"/>
```

```
<xs:element name="E">
```

```
  <xs:complexType>
```

```
    ...
```

```
  </xs:complexType>
```

```
</xs:element>
```

XML Schema

- Using types from other schemas

```
<import namespace="..." schemaLocation="..." />
```

```
<include schemaLocation="..." />
```

- Include uses the same namespace, import uses its own
 - Similar to Java packages

XSD, SOAP encoding

- Simple value → text node
- Compound value (structure, array) → element with child nodes
- Many-to-many relations
 - An index, id attribute must be introduced
 - The association is a text node, its value is the index

XSD, SOAP encoding

- Arrays

- XSD: array index attribute

```
<myList>  
  <element id="1">...</element>  
  <element id="1">...</element>  
</myList>
```

- SOAP: array dimensions and type in the array node

```
<myList soapenc:itemType="element"  
  soapenc:arraySize="2">  
  <item>...</item>  
  <item>...</item>  
</myList>
```

XSD, SOAP encoding

- Multidimensional arrays
 - XSD: multiple parent child relations
 - SOAP:

```
<myList soapenc:itemType="element"  
soapenc:arraySize="2 3">
```

XML, SOAP encoding

- Binary data is a problem as XML is character based
- Solutions
 - Base64 encoding, this results in big XML documents poor for parsing
 - MIME attachments and Content-ID

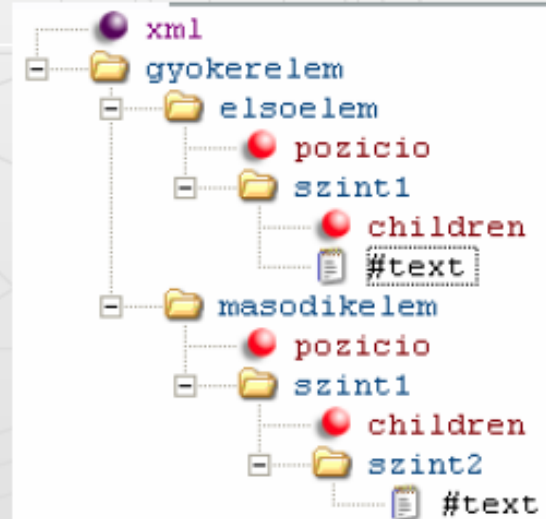
XML parsing

- Restore the complex data structure at the receiver and check the correctness of the received data
- Parser types:
 - DOM
 - SAX

XML parsing

- DOM parsers
 - Read the XML document fully
 - Build a Document Object Model tree

```
<?xml version="1.0" encoding="UTF-8"?>
<gyokerelem>
  <elsoelem pozicio="1">
    <szint1 children="0">Ez a ...
      elemek</szint1>
    </elsoelem>
  <masodikelem pozicio="2">
    <szint1 children="1">
      <szint2>Ez a ...
        elemek</szint2>
      </szint1>
    </masodikelem>
  </gyokerelem>
```



XML parsing

- SAX parsers, only Java standard
 - Reads document as a stream
 - Event based: generates an event if a tag is detected
 - Events: start of an element, end of an element, start document, end document, text node

XML parsing

```
class MyHandler extends DefaultHandler {
    boolean elem;

    @Override
    public void startElement(String uri, String local, String qname,
Attributes attrs) {
        System.out.println("Starting "+qname);
        if (local.equals("elem")) {
            elem= true;
        }
    }

    @Override
    public void endElement(String uri, String local, String qname) {
        System.out.println("End "+qname);
        if (local.equals("elem")) {
            // ...
            elem= false;
        }
    }

    public void characters(char []c, int start, int len) {
        String s = new String(c, start, len);
        System.out.println(s);
    }
}
```

XML parsing

- Pros and cons
 - DOM is good choice for
 - modifying XML documents
 - DOM is bad choice for
 - reading large XML documents – memory needs
 - SAX is good choice for
 - reading XML documents
 - SAX is bad choice for
 - manipulating larger XML documents, though it can be used for smaller ones

Other data serialization languages

- JSON: JavaScript Object Notation
 - Very simple, less overhead
 - Hash tree: string key, string value
 - No semantics, no validation
- YAML: Yet Another Markup Language
 - Very simple, less overhead
 - Hash tree: string key, string/integer/real/bool value
 - Can handle lists
 - No semantics, no validation